CO R E Y K I D D



PYTHON PROGRAMMING

FOR BEGINNERS

A STEP-BY-STEP GUIDE TO LEARNING THE BASICS OF COMPUTER PROGRAMMING AND PYTHON COMPUTER LANGUAGE

Python Programming for Beginners

A Step-by-Step Guide to Learning the Basics of Computer Programming and Python Computer Language

Copyright © 2015 Corey Kidd - All rights reserved

This document is geared towards providing exact and reliable information in regards to the topic and issue covered. The publication is sold with the idea that the publisher is not required to render accounting, officially permitted, or otherwise, qualified services. If advice is necessary, legal or professional, a practiced individual in the profession should be ordered.

From a Declaration of Principles which was accepted and approved equally by a Committee of the American Bar Association and a Committee of Publishers and Associations.

In no way is it legal to reproduce, duplicate, or transmit any part of this document in either electronic means or in printed format. Recording of this publication is strictly prohibited and any storage of this document is not allowed unless with written permission from the publisher. All rights reserved.

The information provided herein is stated to be truthful and consistent, in that any liability, in terms of inattention or otherwise, by any usage or abuse of any policies, processes, or directions contained within is the solitary and utter responsibility of the recipient reader. Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

Respective authors own all copyrights not held by the publisher.

The information herein is offered for informational purposes solely, and is universal as so. The presentation of the information is without contract or any type of guarantee assurance.

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Table of Contents

Introduction

Thank you for purchasing this book and welcome to the wonderful world of computer programming! If this is your first step into computer programming then you have made a great choice in buying this book. Within these pages we will get you up and running with a brilliant programming language that you can use to create great programs.

Before we move into learning the language, lets learn a bit more about what Python is and why you should learn how to use it.

What Is Python

Python is high-level programming language which was created in the late 1980's by Guido van Rossum. It was named as a homage to the world famous British comedy group Monty Python. Python places huge emphasis on simplicity and code readability, making it easy to create applications quickly. This ease and speed makes Python a brilliant language for you to start off with.

High-level programming languages allow you to write computer programs in a language closer to human languages than computer code. Other high-level programming languages include C, BASIC and Pascal. In order to run programs developed in high-level programming languages need to be converted into machine code. To run and test Python games you will need to install the Python interpreter, which does that for you.

When you come to release your program to the world you can find many tools to package up your program into a stand-alone file. This means that you people can run your programs without installing the Python interpreter on your computer.

Why Should You Learn Python

As we mentioned in the last section, there are a lot of high-level programming languages. If you were old enough to have owned a Commodore 64 then you'll remember that BASIC used to come packaged with the computer itself. If you have researched game development then you'll also know that many people use C++ and Java to develop games.

It is often very difficult to decide which language to start with because there is so much choice out there. The good news is that all of the languages are very, very similar to each other. If you have used BASIC or C in the past then you will recognize many of the statements in Python and will many parts of it incredibly familiar. The differences in the languages are within the syntax (structure of statements) and the choice of libraries (pre-written codes and resources to extend the functionality) available.

Once you learn one language really well, it is easier to learn other languages quickly. When it comes to learning your first language you need to find a language which is powerful and easy to learn. You don't want to learn one language then find out that it is unable to produce the programs you want it to.

Python is a brilliant language to start off with because it is both easy to pick up and incredibly powerful. Python is easy to pick up because it is simple, it usually takes a lot less code to develop a program in Python than it would take to develop a similar program in C++. This means that it will be easier for you to learn and quicker for you to develop programs with Python than other programming languages. It also means you are less likely to make mistakes and easier for you to find out where the mistakes are, which is often one of the toughest things for programmers to do.

Python is also incredibly versatile. You can use it for anything from desktop applications to games and you can also develop applications for mobiles too. You can find a lot of libraries which will allow you to extend the functionality of the language itself. So even though the language might not have the initial capability to carry out certain tasks, you will almost always be able to find a library which does exactly what you need.

Finally, Python is cross-platform – which means that you can write your code for one platform and it will work the same on all of the other platforms. This makes things much easier for you if you want to port your program to each of the Windows, Linux and Mac platforms. You won't need to write the code 3 times for 3 different platforms and it also makes testing the program a lot easier too.

Ready to Start?

Sold on Python yet? Excited to get started? You really should be!

Over the next few pages you will set up Python and then will start to program!

Chapter One: Setting Up Python

In this chapter, you will learn:

- How to download and install the Python interpreter
- How to use the Python Shell

Before you do any programming you need to get Python set up on your computer. In this book we will teach you how to program in Python 3, which is the latest version of the software. If you need to use Python 2 in the future then that is fine, both versions of Python are very similar. If you learn Python 3 then you should have no problem understanding Python 2.

In order to download and install the interpreter for Python you just visit https://www.python.org/downloads/ . It will give you an option of the different versions at the top of the page. You just need to choose the latest Python 3 version, click on it and then download it.

If you need a different version of Python then you just need to scroll down the page and choose the version you want to use.

Once you've downloaded the version of Python you need then all you need to do is install it, then we will be ready to start coding.

Using the Python Shell

We are going to do all of our coding on the IDLE (**I**ntegrated **D**eve**L**opment **E**nvironment) program which comes with Python. IDLE is created to be a very simple IDE and it is something you may have seen before if you use Linux as it is bundled with many Linux programs. IDLE was fully coded in Linux and is created to be very easy for beginners to use.

You launch IDLE the same way that you would launch any other program on your computer. When you start up Python it will present you with the Python Shell, which runs Python in interactive mode. This allows you input a command which it will instantly run before waiting for the next command.

You can start by inputting a few simple commands like the ones below:

```
>>> 2 + 5
```

```
>>> 1<5
```

```
>>> print ('Hello World')
```

Type them into the Python Shell and watch as it will instantly return the result of your command. The first command will run the sum 2 plus 5, which will return the answer 7. The second command will run the equation “is 1 less than 5” which will return the answer True. The third command will tell the shell to display the words “Hello World” which it will instantly do.

The Python Shell is perfect for testing out new commands and seeing how Python works, but it will not save any of your coding. If you want to develop a program you need to write your code and save it in a .py file, which is called a Python script. To do this you just need to click on File > New File at the top menu which will bring up the editor we are going to use to develop our programs

Now we're all ready to start coding, in the next chapter we're going to create our first program!

Chapter Two: Hello World

In this chapter, you will learn:

- How to add comments to your code
- To write and run your first Python Program

The first program that you ever write when you are learning a new language is “Hello World”. We're not going to break that age-old programming book tradition here, so get ready to learn how to program Hello World in Python.

Adding Comments to Your Code

The first thing we are going to learn is how to *comment out* lines of text you type in your programming. When you comment out text you tell the interpreter to ignore whatever you type on the line.

“Why would you want to do that?” I hear you all ask.

For non-programmers, the idea of commenting out lines of text doesn't really make much sense, but it is vital that you get into the habit of commenting out text from the beginning. Years ago I developed a huge program without including any comments whatsoever and after 2 full weeks of development I found a bug in some of the early code I wrote. Can you imagine how difficult it was to go back through all of the code and find the one line which was causing the bug.

If you said “nearly impossible” you would be absolutely right.

Your comments tell you what the following piece of code is going to do, which makes it easier when you need to edit things or you need to find a bug.

In Python you comment out a line by typing a # at the beginning of the line. For the current program you are writing you would want to type:

#Displays the words “Hello World” on the screen

If you have more than one line you want to comment out then you can do it by typing three single quotation marks above the first and below the last line. You would do that in the current program in the following way:

'''

Displays the words

“Hello World”

On the screen

'''

Simple, right? Now let's write our program!

Writing Hello World

Hello World is the most simple program you will ever write, but it is also the first program you will ever write in Python (or ever, if you have never programmed before). So get excited!

All you need to do is type the following words into the editor:

```
#Displays the words “Hello World” on the screen
print (“Hello World”)
```

And your program is ready!

Now you need to click File > Save As and save your file as helloworld.py . Do not forget to add the .py extension! It will not do it automatically for you and if you don't add it then the file will not run.

You can then run the program by pressing F5 or clicking Run > Run Module. You will then see the words “Hello World” displayed on the Python Shell.

And We're Done...

And there we have it. You have now officially wrote your first Python program, give yourself a pat on the back.

But great programs aren't created by simply telling your computer to display words on the screen (unfortunately) you will also need to tell your program to hold data and process data. In the next chapter we will learn how to do that.

Chapter Three: Storing and Processing Data

In this chapter, you will learn:

- What variables are and how to use them
- What operators are and how they work

One of the most important parts of any computer program is the way that it stores data and processes data. This is all of the important work in the background which makes your programs work properly.

To store and process data in your programs you will use variables and operators. If you are a programmer then you will know what these are, but if you're not then you are just about to find out.

What Are Variables?

You will store any data you need to use in your programs by using variables. For example, if you want to store the name of the user in your program you would use a variable to do so. Before you use a variable you need to tell the computer to set aside some space to store the data in. This is called defining a variable. You also need to set a value to your new variable. You do that with this statement:

user_name = 'Richard'

You may be wondering what the equals sign (=) is doing in the above statement. In maths = is used to define where the answer should go and you would probably expect to see it at the end of a question. In programming the equals sign is used to assign a value to the variable listed before it. If you write the following details into your IDLE editor:

#Defines the variable user_name

user_name = “Richard

#Displays the variable user_name on the screen

print (“Name = ”, user_name)

Now save and run the program and you will see exactly how this works.

You can also define more than one variable in one go. So rather than defining the variables on two separate lines like this:

user_name = “Richard”

user_age = 25

You could use the following statement:

user_name, user_age = “Richard”, 25

Which saves space when you are writing your new program.

When you name your variables you must use letters, numbers or underscores and the first character must be a letter. The variables are case sensitive, so you could have both user_age and User_Age... but I wouldn't recommend it! Imagine how complicated it would get.

You are also not allowed to use any variables which already have a function in Python. So print, if and for are all out of the question.

Most programmers will use different naming conventions when it comes to creating variables to make the code easier to read – especially if you are coding with others. The naming conventions I use are underscores, so all of my variables are created with an underscore in between the words. For

example, user_name.

There are other naming conventions but this is the one we will use throughout the book. If you are an experienced programmer than you can feel free to stick to the naming conventions that you have used in the past.

What are yhe Different Basic Data Types

In the last section of of this chapter we used two data types, these were integer and string.

An integer is a whole number with no decimal parts. In the last section this was user_age. If you define a number without a decimal point and without quotation marks it will be an integer. For example, user_age = 25.

A string is simply text. In the last section this was user_name. If you put a number in quotes then it will be defined as a string and will not work as a number. For example, user_age = “25”. We will go through some string operators in the next section.

Finally, there is a third data type called float, which is a number which has decimal parts. We have a float in the last section, but this would be numbers like 2.14 or 0.01. For example, user_age = 25.5.

Using Operators

Going back to assigning variables, it is also possible to assign a variable to the value of another variable. So if you write the following program:

```
x = 0

y = 5

print (“x = ”, x)

x = y

print (“x = ”, x)

print (“y = ”, y)
```

Then save and run the program you will find that the x variable is 0 the first time it is displayed, then has turned into 5 by the time it is displayed the second time. When the y variable is printed it retains the same value even though its value has also been taken up by the x variable.

If you switch the x = y to y = x you will find that the y is then changed to 0 at the end of the program.

Many of the main operators are mathematical equations which you can carry out using your variables. The basic operators you can use are +, -, *, /, //, % and **. These symbols are used for addition, subtraction, multiplication, division, floor division, modulus and exponent.

Most of these are self explanatory, but a few will require some extra explanation:

Floor division is where you divide the numbers then round the answer down to the nearest whole number.

Modulus is where you give the remainder when the two numbers are divided.

Finally, exponent is the first number to the power of the second number. If you were performing 10 ** 3 your answer would be 10 to the power of 3.

I'll give a few examples of these operations below (in these examples x = 10 and y = 3):

Addition:

x + y = 13

Subtraction:

x - y = 7

Multiplication:

x * y = 30

Division:

x / y = 3.3

Floor Division:

x // y = 3

Modulus

x % y = 1

Exponent

x ** y = 1000

You can use these different operators to assign values to different variables. If we take addition as an example you would use the following piece of code:

```
#assigns the answer of x + y to x

x += y
```

So x += y works the same way as x = x + y. This works for all of the operators listed above.

Chapter Four: Playing With Words

In this chapter, you will learn:

- How to work with string variables in Python
- How to format strings variables in Python

In this chapter we will go through some operators for string variables. You will need to use words throughout your program in order to tell the user what to do and to display all of the things your program is doing in the background.

Linking Strings Together

Sometimes you will have two strings which you need to link together during your program. You may use this if you want to take a users name full name and you want to refer to them by their first name, or by “Mr/Miss/Mrs” and their second name, or by their full name.

In this case you will define the users name as follows:

#Defines the users name

user_sex = “Mr”

user_firstname = “David”

user_surname = “Christopher”

You can combine the text by using a + sign in the middle of the words. Something you will find if you use the code print ([user_firstname + user_surname]) will display “DavidChristopher” without any space in between.

You can use this code to put a space in between the words:

print (user_firstname, user_surname)

Formatting Strings

You can use the % operator to format strings. This works by writing your string and adding the % symbol. Then add brackets which include the variables and values which will be added to the string.

We have already defined some variables in the last section which we will use in this example of how to format strings.

set_message = “Welcome to this program %s %s %s”

(user_sex, user_firstname, user_secondname)

print (message)

If you save this script and then run it you will find that it prints out as “Welcome to the program Mr David Christopher”. %s, %s and %s are used as place-holders which are replaced with the variables and values you input in the brackets underneath.

When you format using this method you use %s to signify a string, %d to signify a number and %4.2f to signify a float. The 4 in the %4.2f signifies the total number length and the 2 signifies the decimal places.

Setting String Case

You can also easily set the case of the string when you are displaying it on the screen. For example you can use the code print (user_firstname.upper()) which will display the first name in upper case. If you use the code print (user_firstname.lower()) it will display the first name in lower case.

See what I said about Python being simple?

Chapter Five: Make It Interactive

In this chapter, you will learn:

- How to receive and process input from users
- How to receive and process input from files

A computer program isn't much use if it just simply displays stuff you have wrote into the program. It is important to get input from the user of the program themselves in order to truly turn it into an actual computer program that people will use.

As we said earlier, Python is made to be simple and easy to use. With that in mind, Python chose a very good name for function which receives input from the user.

The Input Function

We're going to go all the way back to the information we put into the program in the second chapter, but this time we're going to ask the user for their name rather than making it up for them. In IDLE open a new file and input the following code:

```
#Ask the user for their name and age

user_name = input("Can I take your name: ")

user_age = input("And your age: ")

#Display the information on the screen

print("Hello! My name is", user_name, "and I am", user_age, "years old.")
```

Now save and run the program and you will notice that it asks you for your name and your age then displays the information to you.

The way the function works is by displaying the string you place in the brackets to the user then storing the information which the user inputs in a string.

Because the information is by default stored in a string, you will need to directly set the information to an integer or float if you wish it to be stored as such. You do that with the following code for an integer:

```
user_age = int(input("And your age: ")
```

And the following code as a float:

```
user_age = float(input("And your age: ")
```

Advanced Print

We know how to display text to our users by using the print function – and we know how to format the text. But there are a few extra advanced print functions you don't know about just yet!

The first advanced print function is to display the message over more than one line using the "triple-quote" symbol, which is simply "" or """. An example of how to use this:

print (""Welcome to this program.

I will need to know your name

on the next page."")

This will print the message over the three lines, which makes it easier to read.

The second advanced print function is to use "escape characters". Escape characters allow you to print characters which are "difficult to type" like tab, new line or a quotation mark. You are able to print these characters by using a \ (backslash) character.

To type a tab you use the code: \t.

For example, print ("Look at this\tTAB!) will display:

look at this	tab!

To type a new line you use the code: \n

For example, print ("Look at this\nNew Line!") will display:

Look at this

New Line!

To type a backslash you use the code: \\

For example, print ("Look at this \\ backslash.") will display:

Look at this \ backslash.

To use a double quote or a single quote you use the code: \" or \'

Which one you need to use an escape character for will depend on whether you use single quotes or double quotes in your text. If you use single quotes then you can display a double quote with no issue, but a single quote will end the string. Vice-versa if you use a single quote.

As an example, print ("Hey, I can display a \" today.") will display

Hey, I can display a " today.

If you want to use the letter after the \ to be counted as a special character you need to add an r before the first quotation mark. For example, print (r“Look at\this.”) will display:

Look at\this.

While, print (“Look at\this.”) will display:

Look at   his.

Receive Input From Files

Receiving input from text files is very easy. First, create an example text file, type out 5 lines of text and save it as exampletext.txt to the same folder you are going to save the program to. To open the text file you simply use the following test:

```
#Opens a file

text_file = open ('exampletext.txt', 'r')
```

This line will open the file named in the brackets and set it to a read-only file with the 'r'. You can open the file for reading ('r), writing ('w'), reading and writing ('r+') or appending ('a'). If you are writing or appending (adding data to the end) to a file which does not exist, your program will create the file.

Once you have opened the file you need to read from the file with the following code:

```
#reads lines from file and displays them on the screen

firstline = text_file.readline()

secondline = text_file.readline()

print (firstline)

print (secondline)
```

When you use the readline() function it will read the next line from your file. Which means that this code will display the first two lines of your file.

Once you have finished reading from the file you want to close the file to save up space in your program. Which you do with this code:

```
text_file.close()
```

If you choose to write to a file, you open and close the files in the same way, but what you do in between is different. If you want to overwrite everything currently contained in a file then you need to use 'w' when you open a file, if you just want to add to the file then you need to use 'a'. You write to the file using the same code whether you are writing or appending a file. Here is the code to write to a file. I have chosen to but you can choose to write if you wish:

```
text_file = open ('exampletext.txt', 'a')

#writes text to the file

text_file.write('\nThis will be added to the end of my file.')

text_file.write('\nAs will this.')
```

text_file.close

You will have noticed the new line escape character \n in the text above, if you fail to add that escape character then the text will run on in the same line. As text is read line by line in Python you want to ensure that you write line by line when you need to.

In the next section we are going to start controlling the flow of your program. Which is when things get very interesting!

Chapter Six: Controlling the Flow of Your Program

In this chapter, you will learn:

- What condition statements are and how they work
- How to use control flow tools in your program

Now we're on to one of the most important (and interesting) chapter in this book. This part of the program is where your program will start to chose which things to do and decide which things not to do.

In this chapter we will learn about control flow tools, which are incredibly important because they control the flow of the program. Without control flow tools the program will just simply run through every piece of code you write from beginning to end.

Condition Statements

A condition statement is a statement which checks a condition and then run a certain part of code depending on whether the condition is met or not.

The different conditions you can check are the ones below:

Are the variables the same?

x == y

Are the variables different?

X != y

Is one variable greater than the other?

x > y

Is one variable greater than or equal to the other?

x >= y

If one variable smaller than or equal to the other?

x <= y

Is one variable smaller than the other?

x < y

The statements will return true or false. You can check more than one statement by placing a logical operator between the statements. Here are the logical operators you can use in Python:

Are the two statements true?

x == y and a == b

Are at least one of the statements true?

x == y or a == b

Is the statement untrue?

not x == y

The If Statement

The if statement is a very useful statement something which is commonly used in many different programs. This statement checks if a statement is true or not and only runs the lines of code if the statement is true.

If you have coded in other programming languages then you will find that the if statement in Python is a little bit different than it is in the other languages. For a start, you don't need to use brackets (), curly brackets {} or the statement End If to define where the statement begins and ends. Instead, Python uses indentation to define the beginning and the end of the if statement instead, which makes it much easier.

The structure of an if statement is:

if first condition is met:

- do A

elif second condition is met:

- do B

else:

- do C

If you have used other programming languages you will probably already have worked out that elif means else if. If you haven't used another programming language, elif will check another condition and run the code if it it is met. You can have as many elif statements as you need.

Finally, else is the code that will run if none of the conditions are true.

Let's run an example to see how the if statement works. Type this into IDLE, then save and run it.

check_input = input(“Choose 1 or 2”)

#check the input and run different code depending on the choice

if check_input == “1”:

- print (“Sorry, you lose!”)

elif check_input == “2”:

- print (“Congratulations! You win!”)

else:

- print (“Your choice is invalid.”)

Now test the program out and run all of the responses. Have a think about the programs you use regularly, how might the if statement be used to make these programs work?

There is another kind of if statement you can use if you only need to do something simple which can fit on one line. This is called the inline if. Here is an example of how this works:

print (“Congratulations! You Win!” if check_input == 2 else “Sorry, you lose!”)

The line above will display “Congratulations! You Win!” if check_input is 2, otherwise it will display “Sorry, you lose!”

Loops

Loops are incredibly important when it comes to programming. As the name suggests, a loop will execute the same code over and over again. This is very useful for running the main part of the program over and over again.

There are different kinds of loops which you can use in Python which we will go through now.

The For Loop

The for loop will run through the code over and over again until the statement is no longer true. For example:

user_message = input (“Type in a message: “)

for a in user_message:

- print(a)

If you type this into IDLE and run the program you will find that the message will be displayed one letter per line.

The for loop is a good place to teach you about lists. Lists are lists of data stored within one variable. You declare a list using the following statement:

set_list = [“Zero”, “One”, “Two”, “Three”, “Four”, “Five]

If you then write the code print (set_list) it will display the entire list on one line. You can also choose to display one part of the list by writing the code print (set_list[3]). The number in the square brackets is the number in the list you want to display.

You can use the for loop to run through a list one by one with the following code:

for set_list_print in set_list:

- print (set_list_print)

This loop will then display each piece of data in the list one by one on each line.

The While Loop

The while loop repeats the loop while a condition is true. For example:

set_number = input(“Type in a number between 5 and 20)

#Run loop as long as the set_number is more than 0

while set_number > 0:

- print (“Number: ”, set_number)
- set_number -= 1

Now save and run the program and see what happens.

Whenever you are running a while loop, make sure that the loop is able to end at some point otherwise it will loop forever until you somehow close the program down!

Break and Continue

In some programs you may want to find a way to exit the loop or to skip past a part of the loop.

If you wish to exit the loop you need to use the word "break". Once the program reads the break it will come out of the loop and carry on with the program. For example:

set_list = ["Zero", "One", "Two", "Three", "Four", "Five]

num = 0

for set_list_print in set_list:

- print (num, set_list_print)
- num += 1
- if num == 3
- break

Run the program and see what happens. You will find that the program stops once the num variable hits 3, even though it would otherwise have ran until the num variable reached 5.

If you want to skip part of the loop then you use the variable "continue", which will ignore the next piece of code and continue on with the loop. For example:

set_list = ["Zero", "One", "Two", "Three", "Four", "Five]

num = 0

for set_list_print in set_list:

- print (num, set_list_print)
- if num == 3
- continue
- num += 1

Edit the code you typed out above with this piece of code and then run it to see what happens. You will find that the num variable only reaches 3 and then stays at the same value for the rest of the program.

Chapter Seven: Creating Functions and Importing Modules

In this chapter, you will learn:

- What functions are and how to create them
- What modules are and how to import them into your program

In previous chapters we have used functions but not explained what they are. Functions are basically pieces of code which have already been written, meaning that you don't need to rewrite the full piece of code every time you want to do something.

Python has many pre-written functions ready for you which are saved in Modules that you can add into your programs. You can find many modules available for you online, which can speed up the development in your next program.

Let's start by explaining how to importing new modules.

Importing New Modules

Importing modules is incredibly easy. You simply use the following statement:

import module_name

So, for example, if you want to use the date and time in Python you need to import the datetime module. You do that with the following statement:

import datetime

Once you have imported datetime you can use all of the functions from the datetime module. You can use the date function from the datetime module in the following way:

datetime.date (2015, 9, 21)

The function above sets the date in the program to the 21st of September 2015. You can save yourself a lot of typing by importing the datetime module in the following way:

import datetime as d

This means that you can call any function from the datetime module by using d rather than datetime. For example:

d.date (2015, 9, 21)

If you want to create your modules then all you need to do is write the functions (we'll teach that in the next section). You then save the file as a .py file and save it in the same folder as the program you want to use it in. If you download a module you have found online, you also save it in the same folder as the program you want to use it in.

You then import it in the same way that you import any other module. If you have saved a module called checknumber.py then you would import it by typing out:

import checknumber

Creating Your Own Functions

Creating functions in Python is called “defining functions” and it isn't difficult to do. You define a function in the following way:

def function_name(parameters)

- code
- return[expression]

Once you have defined the function you type the code out and end the function with the word return.

Here is an example of a function:

def check_if_true(number_check)

- if number_check == 5:
    - return true
- else:
    - return false

To use this function you could use the following code:

check_number = input(“Enter a number”)

answer = number_check(check_number)

- print answer

If you run the code then you will find that if you type the number 5 it will display “true” and if not it displays “false”. This works because the return statement sends true or false back to the variable “answer”.

It is important to recognise that any variables declared inside a function can only be used inside the function. These are called local variables. Variables declared outside of functions can be used inside of a function as well, these functions are called global variables.

You can declare a variable in a function with the same name as a global variable, but it isn't recommended as it can get confusing. If you do declare a variable outside and inside of a function with the same name then the only variable used inside the function will be the local variable.

Best Practices & Common Mistakes

Do's

- Use Comments

It is important to use comments as much as possible in your work. Get into the habit right from the beginning. Even though your smaller projects may not require much editing, it will really come in handy when you are creating big projects.

- Stick to Your Naming Conventions

Name your variables in order with your chosen naming convention. It helps you to avoid silly mistakes. Again, this might not seem important when you are creating small programs. But when you start creating big programs you need to do whatever you can do to help avoid mistakes.

- Save Your Program Regularly

I have once written hundreds of lines of code without saving and lost it all with a computer crash. I can't tell you how sick I felt afterwards. Don't make my mistake! Save your program as often as possible and you won't risk losing it.

- Keep a Backup

Make a backup of your program and before you make any big changes to your program, save it to your backup. Sometimes your changes won't work very well and they may even cause awful bugs in your program. In those cases you will be relieved to be able to go back to the way it was before you made those changes.

Don’ts

- Forget to Place a : On If Statements

One big mistake most newbies to forget to place a : at the end of the line on if statements. If you're getting errors and you don't know what they are, make sure you've remembered to place a : at the end of the line.

- Use The Wrong Variable Scope

Remember that any variable declared in a function can only be used in the function. If you try to use it outside of the function it won't work.

- Cause an Infinite Loop!

If you're using a while loop, don't forget to make sure that the loop can end at some point!

Conclusion

Well done for working your way through this book and learning how to program in Python. You have learned everything you need to learn in this book to start programming in Python. In fact, you have already created many programs.

Now that you have learned how to create computer programs in Python you can start your journey into the wonderful world of computer programming. Use this book as a reference whenever you get stuck with anything, which you might find you do moving forward.

Now that you have all of the basics of Python you can start creating your own programs. I'm excited to see what you can produce – and you should be very excited too!